In the following examples, *ITALIC* denotes text duplicated within the example, ~~STRIKEOUT~~ denotes text removed from the previous example, and UNDERLINE denotes text that is new or changed from the previous example.

Start with the callee and caller in F77:

```
                                    SUBROUTINE S
FUNCTION F(X)
COMPLEX F                           COMPLEX F
REAL X
F=...
END FUNCTION
                                    C=F(Y)
                                    END SUBROUTINE
```

5 New procedure attributes and facilities in F90 lead to the creation of the interface block:

```
                                    SUBROUTINE S
                                    ~~COMPLEX F~~
                                    INTERFACE
FUNCTION F(X)                       FUNCTION F(X)
COMPLEX F                           COMPLEX F
REAL,OPTIONAL :: X                  REAL,OPTIONAL :: X
F=...                               ~~F=...~~
END FUNCTION                        END FUNCTION
                                    END INTERFACE
                                    C=F(Y)
                                    END SUBROUTINE
```

Little difference for dummy procedures if actual is external procedure:

```
                                    SUBROUTINE S(D)
                                    INTERFACE
FUNCTION F(X)                       FUNCTION D(X)
COMPLEX F                           COMPLEX D
REAL,OPTIONAL :: X                  REAL,OPTIONAL :: X
F=...                               END FUNCTION
END FUNCTION                        END INTERFACE
                                    C=D(Y)
                                    END SUBROUTINE
```

Problem when actual is a module procedure. Original F90 “solution” is recursive USE of module.

```
MODULE M
TYPE T; ... ; END TYPE
                                    SUBROUTINE S(D)
                                    INTERFACE
FUNCTION F(X)                       FUNCTION D(X)
                                    USE M
TYPE(T) :: F                        TYPE(T) :: D
REAL,OPTIONAL :: X                  REAL,OPTIONAL :: X
F=...                               END FUNCTION
END FUNCTION                        END INTERFACE
                                    C=D(Y)
                                    END SUBROUTINE
END MODULE
```

That "solution" has been interpreted as not legal. Current F2K is solution is IMPORT:

```
MODULE M
TYPE T; ... ; END TYPE
                                        SUBROUTINE S(D)
                                        INTERFACE
FUNCTION F(X)                           FUNCTION D(X)
                                        IMPORT T
TYPE(T) :: F                            TYPE(T) :: D
REAL,OPTIONAL :: X                      REAL,OPTIONAL :: X
F=...                                   END FUNCTION
END FUNCTION                            END INTERFACE
                                        C=D(Y)
                                        END SUBROUTINE
END MODULE
```

1) This put the IMPORT in the middle of text copied from F. (complicates text editing)

2) D gets T from S rather than M (as F does), so any declaration of the name T in S could cause D to end up wrong. (Nothing else in S has that effect.)

3) Some people object to the necessity of specifically identifying T as imported into D, given that you don't have to do it for F.

The proposed CONTEXT statement addresses these points:

```
MODULE M
TYPE T; ... ; END TYPE
                                        SUBROUTINE S(D)
                                        INTERFACE
                                        CONTEXT MODULE
FUNCTION F(X)                           FUNCTION D(X)
                                        ~~IMPORT T~~
TYPE(T) :: F                            TYPE(T) :: D
REAL,OPTIONAL :: X                      REAL,OPTIONAL :: X
F=...                                   END FUNCTION
END FUNCTION                            END INTERFACE
                                        C=D(Y)
                                        END SUBROUTINE
END MODULE
```

1) CONTEXT MODULE is outside the text of the interface body.

2) CONTEXT MODULE is defined to go directly to M, not through S.

3) Since this is now host association into M (just like F), T no longer needs to be named explicitly.

01-397 also has CONTEXT EXTERNAL to allow explicit statement of the existing default. The Enhanced Modules TR could introduce CONTEXT SUBMODULE *[name]*.

Notes (from the discussion):

1) We might spell CONTEXT MODULE as IMPORT(MODULE) or IMPORT(M).

2) The existing IMPORT might be redefined to import from M.

3) Some people like having T named in the import list. Presumably, they would like it for F as well as for D. Allow similar import control after the CONTAINS? (But make specifying the list optional!)

•